Authentication Bypass

**Task 1 - Brief**

In this room, we will learn about different ways website authentication methods can be bypassed, defeated or broken. These vulnerabilities can be some of the most critical as it often ends in leaks of customers personal data.

**Task 2 – Username Enumeration**

A helpful exercise to complete when trying to find authentication vulnerabilities is creating a list of valid usernames, which we'll use later in other tasks.

If you try entering the username admin and fill in the other form fields with fake information, you'll see we get the error An account with this username already exists. We can use the existence of this error message to produce a list of valid usernames already signed up on the system by using the ffuf tool below. The ffuf tool uses a list of commonly used usernames to check against for any matches.

Command - ffuf -w /usr/share/wordlists/SecLists/Usernames/Names/names.txt -X POST -d "username=FUZZ&email=x&password=x" -H "Content-Type: application/x-www-form-urlencoded" -u http://10.10.175.197/customers/signup -mr "An account with this username already exists"

In above example:

-w, - argument selects the file's location on the computer that contains the list of usernames that we're going to check exists.

-X : argument specifies the request method, this will be a GET request by default, but it is a POST request in our example.

-d: argument specifies the data that we are going to send. In our example, we have the fields username, email, password and cpassword.

We've set the value of the username to FUZZ. In the ffuf tool, the FUZZ keyword signifies where the contents from our wordlist will be inserted in the request

-H: argument is used for adding additional headers to the request. In this instance,

We're setting the Content-Type to the webserver knows we are sending form data.

-u: argument specifies the URL we are making the request to,

And Finally -

-mr: argument is the text on the page we are looking for to validate we've found a valid username

Save the results from this command into a file called valid\_usernames.txt which we can use in a later task and then answer the questions below.

**Answer the question:**

1.What is the username starting with si\*\*\* ?

2. What is the username starting with st\*\*\* ?

3. What is the username starting with ro\*\*\*\* ?

Command : ffuf -w /usr/share/wordlists/SecLists/Usernames/Names/names.txt -X POST -d "username=FUZZ&email=x&password=x&cpassword=x" -H "Content-Type: application/x-www-form-urlencoded" -u http://10.10.146.235/customers/signup -mr "username already exists" -o valid\_username.txt

Result:

admin [Status: 200, Size: 3720, Words: 992, Lines: 77]

robert [Status: 200, Size: 3720, Words: 992, Lines: 77]

simon [Status: 200, Size: 3720, Words: 992, Lines: 77]

steve [Status: 200, Size: 3720, Words: 992, Lines: 77]

:: Progress: [10177/10177] :: Job [1/1] :: 141 req/sec :: Duration: [0:01:13] :: Errors: 0 ::

**Task 3 – Brute Force**

Using the valid\_usernames.txt file we generated in the previous task, we can now use this to attempt a brute force attack on the login page (http://10.10.146.235/customers/login).

A brute force attack is an automated process that tries a list of commonly used passwords against either a single username or, like in our case, a list of usernames.

When running this command, make sure the terminal is in the same directory as the valid\_usernames.txt file.

Command:

user@tryhackme$ ffuf -w valid\_usernames.txt:W1,/usr/share/wordlists/SecLists/Passwords/Common-Credentials/10-million-password-list-top-100.txt:W2 -X POST -d "username=W1&password=W2" -H "Content-Type: application/x-www-form-urlencoded" -u http://MACHINE\_IP/customers/login -fc 200

This ffuf command is a little different to the previous one in Task 2. Previously we used the FUZZ keyword to select where in the request the data from the wordlists would be inserted, but because we're using multiple wordlists, we have to specify our own FUZZ keyword. In this instance, we've chosen

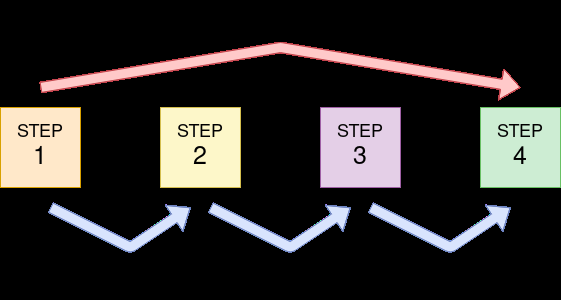
W1 - for our list of valid usernames and

W2 - for the list of passwords we will try. The multiple wordlists are again specified with the -w argument but separated with a comma. For a positive match, we're using the -fc argument to check for an HTTP status code other than 200.

If file not work then make another file with nano and put all username and run again with this file.

**Task 4 – Logic Flaw**

Sometimes authentication processes contain logic flaws. A logic flaw is when the typical logical path of an application is either bypassed, circumvented or manipulated by a hacker. Logic flaws can exist in any area of a website, but we're going to concentrate on examples relating to authentication in this instance.



Logic Flaw Example:

The below mock code example checks to see whether the start of the path the client is visiting begins with /admin and if so, then further checks are made to see whether the client is, in fact, an admin. If the page doesn't begin with /admin, the page is shown to the client.

if( url.substr(0,6) === '/admin') {

# Code to check user is an admin

} else {

# View Page

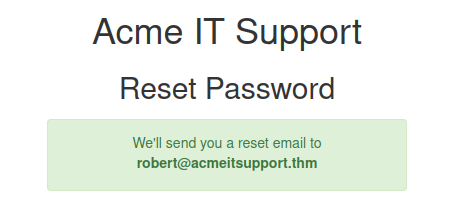
}

Because the above PHP code example uses three equals signs (===), it's looking for an exact match on the string, including the same letter casing. The code presents a logic flaw because an unauthenticated user requesting /adMin will not have their privileges checked and have the page displayed to them, totally bypassing the authentication checks.

Logic Flaw Practice

We're going to examine the Reset Password function of the Acme IT Support website (http://10.10.146.235/customers/reset). We see a form asking for the email address associated with the account on which we wish to perform the password reset. If an invalid email is entered, you'll receive the error message "**Account not found from supplied email address".**

For demonstration purposes, we'll use the email address robert@acmeitsupport.thm which is accepted. We're then presented with the next stage of the form, which asks for the username associated with this login email address. If we enter robert as the username and press the Check Username button, you'll be presented with a confirmation message that a password reset email will be sent to robert@acmeitsupport.thm.



At this stage, you may be wondering what the vulnerability could be in this application as you have to know both the email and username and then the password link is sent to the email address of the account owner.

This walkthrough will require running both of the below Curl Requests on the AttackBox which can be opened by using the Blue Button Above

In the second step of the reset email process, the username is submitted in a POST field to the web server, and the email address is sent in the query string request as a GET field.

Let's illustrate this by using the curl tool to manually make the request to the webserver.

*Command: curl 'http://10.10.146.235/customers/reset?email=robert%40acmeitsupport.thm' -H 'Content-Type: application/x-www-form-urlencoded' -d 'username=robert'*